On parallel pre-conditioners for pressure Poisson equation in LES of Complex Geometry Flows

K. M. Singh[a], E. J. Avital[b], J. J. R. Williams[b], C. Ji[c], X. Bai[b] and A. Munjiza[b]

[a]Department of Mechanical and Industrial Engg., IIT-Roorkee, Roorkee, India
[b]School of Engineering and Materials, Queen Mary University of London, London, UK
[c]State Key Lab. of Hydraulic Engg. Simulation and Safety, Tianjin University, Tianjin, China

Corresponding Author

Dr. E. J. Avital
School of Engineering and Materials Science
Queen Mary, University of London
Mile End Road
London, E1 4NS, UK
E-mail: e.j.avital@qmul.ac.uk
Phone: +44 (0)20 7882 3616
Fax: +44 (0)20 8983-1007

ABSTRACT
This paper presents an assessment of fast parallel pre-conditioners for numerical solution of the pressure Poisson equation arising in large eddy simulation of turbulent incompressible flows. Focus is primarily on the pre-conditioners suitable for domain decomposition based parallel implementation of finite volume solver on non-uniform structured Cartesian grids. Bi-conjugate gradient stabilized (BICGSTAB) method has been adopted as the Krylov solver for the linear algebraic system resulting from the discretization of the pressure Poisson equation. We explore the performance of multigrid pre-conditioner for the non-uniform grid and compare its performance with additive Schwarz pre-conditioner, Jacobi and SOR(k) pre-conditioners. Numerical experiments have been performed to assess the suitability of these pre-conditioners for a wide range of non-uniformity (stretching) of the grid in the context of LES of a typical flow problem. It is seen that the multigrid preconditioner shows the best performance. Further, the SOR(k) preconditioner emerges as the next best alternative.

Keywords: Poisson equation; BiCGSGTAB; Domain decomposition; Preconditioners; Multigrid

1. Introduction

1.1 Background

Numerical solution of incompressible Navier-Stokes equations invariably requires solution of an elliptic equation for pressure (called the pressure Poisson equation) to enforce continuity. In case of large eddy simulation (LES) or direct numerical simulation (DNS) of turbulent flow, the time step required for solution accuracy is usually smaller than that enforced by stability requirements of explicit methods. Hence, explicit time integration schemes of Adams-Bashforth and Runge-Kutta family are preferred in LES/DNS. Thus, in such simulations, the most expensive part is the solution of the pressure Poisson equation required at each time step [1]. Further, large number of grid points required in LES/DNS invariably requires use of parallel computers. Integration of momentum equations with explicit time integrations schemes on parallel architectures is rather straightforward. However, parallel solution of the pressure Poisson equation is not that straightforward. Therefore, substantial research effort has been devoted to the development of fast parallel Poisson solvers in recent decades [2–8] .

Application of a discretization scheme (such as finite difference, finite volume or finite element) to the pressure Poisson equation leads to a sparse linear system. Nature of this system depends on the discretization scheme, the underlying grid and the boundary conditions. Cell-centred finite difference or finite volume discretization usually leads to a symmetric and positive definite linear system on uniform Cartesian grids. On the other hand, the resulting system would be indefinite on non-uniform Cartesian grids. With finite element discretization, the resulting system would be usually symmetric. The nature of the resulting linear system dictates the choice of the numerical scheme for its numerical solution. Further, in turbulent flow simulations, the size of the system would be very large which dictates the use of iterative solvers. Basic iterative solvers such as Jacobi/Gauss-Seidel are easy to program but have very slow convergence. Hence, in practical applications, Krylov subspace methods [9] or multi-level multigrid methods are normally used [10,11].

Multigrid methods can be used either as stand-alone solvers [12–15] or with Krylov subspace acceleration [9]. Further, multigrid methods can be based on the problem and underlying grid (the geometric multigrid, GMG) [10] or can be based entirely on the linear system with no reference to what led to this linear system (algebraic multigrid, AMG) [16]. On structured Cartesian grids where a sequence of nested grids can be easily generated, the geometric multigrid scheme

provides an optimal performance, and can be easily parallelized [10]. It has also been extended for use on complex problem domains in conjunction with fictitious domain or immersed boundary method [17]. Algebraic multigrid method is very promising for unstructured grids on complex geometries [16]. However, its parallel implementation is much more involved, and it has substantial overheads in terms of additional memory and set-up time requirements as compared to the geometric multigrid method.

Krylov subspace methods such as conjugate gradient, GMRES or BiCGSTAB are robust iterative solvers. However, their performance is critically dependent on the choice of the pre-conditioner. Most of the preconditioning techniques for Krylov subspace methods can be broadly put in two categories: (a) incomplete factorization such as incomplete Cholesky (IC) or incomplete LU (ILU) based preconditioners, and (b) sparse approximate inverse [18]. For full details of the recent developments, see Saad [9] and the recent review of Benzi [18]. It may be noted that incomplete factorization based pre-conditioners are more difficult to parallelize as compared to sparse approximate inverse preconditioners. In parallel implementations, yet another category of preconditioners based on domain decomposition have been developed [9,19]. These are usually based on some variant of additive or multiplicative Schwarz methods. Multigrid methods (geometric or algebraic) can also be thought of as multiplicative multilevel preconditioners of this family [19]. In context of sparse inverse approximations, a heuristic incomplete Poisson preconditioner has been proposed by Ament et al. [20] for GPU based parallel implementation. Recent developments with Krylov subspace methods include use of deflation techniques with PCG [3,21] and GMRES [22] and BiCG [23,24].

On unstructured grids, use of purely algebraic methods for design of a preconditioner is eminently understandable. Parallel version of these preconditioners (whether based on incomplete factorization, sparse approximate inverse or algebraic multigrid) have been developed in conjunction with graph-partitioning techniques [18]. However, in the context of structured Cartesian grids, it is desirable to design and use preconditioners which exploit the information about the underlying grid and the PDE to the extent possible. Though this latter approach leads to a problem dependent method, it is also likely to lead to an optimal set of methods for a given application. Advantages of this approach have already been observed in case of numerical simulations based on uniform Cartesian grids irrespective of the complexity of the problem domain which can be overcome using fictitious domain or immersed boundary approach [17]. The present work is another attempt in this direction but with non-uniform Cartesian grids which can better capture the development of shear driven flows.

1.2 Motivation and Scope

For flow problems in complex geometry, unstructured grid finite volume methods have enjoyed the widest application in CFD analysis, especially in commercial CFD codes. However, with recent developments in immersed boundary methods [25–31], structured Cartesian grid methods are back in favour. These methods have been extended for a wide variety of flow problems including moving boundary and fluid structure interaction problems [32–35]. Cartesian grid methods are very easy to parallelize using domain decomposition, and hence, these are especially suited for LES/DNS using massively parallel computers. In view of these features, the authors' research group has been involved in the development of a complex geometry large eddy simulation code, CgLES based on Cartesian grids and domain decomposition approach. This code was initially developed based on staircase approximation of curved boundary surfaces on a uniform Cartesian grid and use of SOR as pressure solver. It was augmented with a fictitious domain multigrid preconditioner along with PCG method as Poisson solver. This development significantly enhanced the parallel scalability and efficiency of this code [17]. With incorporation of immersed boundary method, CgLES has been used to solve a wide variety of flow problems [36–38]. To further improve its efficacy, provision of non-uniform grids with arbitrary gradation has been added. The aim of the present research is to provide a set of efficient parallel Poisson solvers in the domain decomposition framework.

Finite volume discretization on a non-uniform grid leads to an indefinite linear system. Given the domain decomposition based framework, we have opted for BICGSTAB as the Krylov subspace solver. The method is very robust for general sparse linear systems, and is much easier to implement than GMRES in a parallel program. The objective of the present research is to develop and implement a set of parallel preconditioners which can exploit the geometric information as much as possible, and can be implemented with minimal additions to the existing data structure. The possible candidates are additive Schwarz, SOR(k) and multigrid preconditioners [9]. In development of multigrid preconditioners on arbitrarily graded non-uniform Cartesian grid, there are issues associated with inter-grid transfer operators (prolongation and restriction operators) which must be addressed.

With the preceding objective, we summarize the plan of the remaining of the paper. In the next section, we provide a brief overview of the governing equations, discretization and parallel implementation. This is followed by the BICGSTAB algorithm, details of the preconditioners and their parallel implementation, numerical results and conclusions in the succeeding sections.

2. Problem Statement: Governing Equations and Discretization

2.1 Governing Equations

Governing equations for unsteady incompressible flow are the continuity and momentum equations given by

$$\frac{\partial v_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial(\rho v_i)}{\partial t} + \frac{\partial(\rho v_i v_j)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} \tag{2}$$

Preceding equations are solved as such in direct numerical simulation (DNS) of turbulent flows. In large eddy simulation (LES), these equations must be filtered to obtain the pertinent equations. However, the filtered continuity and momentum equations have the same form as that of Eq. (1) and Eq. (2) if we assume that **v** and p represent the filtered velocity and pressure fields, and **τ** represents the sum of viscous and subgrid scale (SGS) stresses.

2.2 Discretization of Navier-Stokes Equations

In LES/DNS of turbulent flows, accuracy considerations dictate use of fairly small values of time step in temporal integration. Thus, the time step is small enough to satisfy the stability requirements of explicit time integration methods. Hence, explicit techniques of the Adams-Bashforth and the Runge-Kutta family are very popular in LES/DNS. For simplicity of representation, let us consider discretization using the explicit Euler method. Further, use of finite difference or finite volume discretization on a structured grid leads to the following set of discrete equations in the context of projection methods [1]:

$$\frac{(\rho v_i)^* - (\rho v_i)^n}{\Delta t} = -\frac{\delta(\rho v_i v_j)^n}{\delta x_j} + \frac{\delta \tau_{ij}^n}{\delta x_j} \tag{3}$$

$$(\rho v_i)^{n+1} = (\rho v_i)^* - \Delta t \frac{\delta p^{n+1}}{\delta x_i} \tag{4}$$

$$\boxed{\frac{\delta}{\delta x_i}\left(\frac{\delta p^{n+1}}{\delta x_i}\right) = \frac{1}{\Delta t}\frac{\delta(\rho v_i)^*}{\delta x_i}} \tag{5}$$

where superscripts n and n+1 denote time levels, superscript * denotes an intermediate velocity field and δ denotes the discrete spatial discretization operator. Equations (3) and (4) represent

explicit formulae for evaluation of the unknown quantities in terms of already computed field variables. In contrast, Eq. (5) represents a discrete Poisson equation which must be solved at each time step before Eq. (4) can be used to obtain a divergence-free velocity field.

On a non-uniform cell-centred Cartesian grid, collection of discrete equations (5) for all the computational nodes leads to an indefinite sparse linear system of equations

$$\mathbf{Ax = b} \tag{6}$$

which must be solved using a suitable solver. Choice of the solver must account for the indefinite nature of the system matrix and the ease of parallel implementation in a domain decomposition framework. Most commonly used Krylov subspace solvers for indefinite systems are GMRES, bi-conjugate gradient (BiCG) and bi-conjugate gradient stabilized (BICGSTAB) methods. Amongst these methods, BICGSTAB method is very robust and is the easiest to implement in parallel. Hence, we have opted for this method for the solution of Eq. (6) in this work.

2.3 Parallel Implementation

Given the large number of grid points involved in LES/DNS, it is not possible to perform the numerical simulations on a single serial machine. Hence, a simulation code must be developed which can exploit large scale high performance parallel clusters. In the context of structured grids, the best approach for parallelization is the domain decomposition wherein the problem domain is decomposed into a collection of sub-domains (blocks). Each block is implemented as a data structure which contains the geometric, grid and algorithm specific data for its partition of the problem domain plus an overlap region (of one grid layer in context of second order central difference discretization). Set of one or more blocks can be mapped to one processor core which takes care of all the computations involved in numerical integration represented by Eqs. (3-5) for these blocks [17]. In this domain-decomposition framework, implementation of the projection and correction steps represented by Eqs. (3) and (4) is straightforward. Complicated part is the implementation of the pressure Poisson solver which depends on the choice of the linear algebraic solver for Eq. (6), and would be discussed latter.

3. Biconjugate Gradient Stabilized (BICGSTAB) Method

3.1 BICGSTAB Algorithm

BICGSTAB algorithm is a transpose-free method which does not require matrix-vector products involving transpose of system matrix **A**. The preconditioned version of the algorithm for generic linear system of equations **Ax** = **b** requires two calls to the preconditioner. Let $(\mathbf{u}, \mathbf{v})$ denote the inner product of vectors **u** and **v**. With preconditioning matrix denoted as **M**, the BICGSTAB algorithm can be expressed as follows:

- *Starting with an initial guess* $\mathbf{x}_0$, *compute* residual vector $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. *Choose a dual vector* $\mathbf{r}_0^*$ *such that* $(\mathbf{r}_0^*, \mathbf{r}_0) \neq 0$, *say* $\mathbf{r}_0^* = \mathbf{r}_0$. *Set* $\rho_{-1} = \alpha_{-1} = \omega_{-1} = 1, \ \mathbf{v}_{-1} = \mathbf{p}_{-1} = \mathbf{0}$
- *For i = 0, 1, 2, …. until convergence DO*
  1. $\rho_i = (\mathbf{r}_i, \mathbf{r}_0^*), \quad \beta_{i-1} = \left(\frac{\rho_i}{\rho_{i-1}}\right)\left(\frac{\alpha_{i-1}}{\omega_{i-1}}\right)$
  2. *If* (*i* == 0) $\mathbf{p}_i = \mathbf{r}_i$
     Else $\mathbf{p}_i = \mathbf{r}_i + \beta_{i-1}(\mathbf{p}_{i-1} - \omega_{i-1}\mathbf{v}_{i-1})$
  3. *Obtain* $\hat{\mathbf{p}}$ *by solving* $\mathbf{M}\hat{\mathbf{p}} = \mathbf{p}_i$ (First call to the preconditioner)
  4. $\mathbf{v}_i = \mathbf{A}\hat{\mathbf{p}}, \ \alpha_i = \rho_i / (\mathbf{v}_i, \mathbf{r}_0^*), \ \mathbf{s}_i = \mathbf{r}_i - \alpha_i \mathbf{v}_i,$
  5. *Obtain* $\mathbf{z}_i$ *by solving* $\mathbf{M}\mathbf{z}_i = \mathbf{s}_i$ (Second call to the preconditioner)
  6. $\mathbf{t}_i = \mathbf{A}\mathbf{z}_i, \ \omega_i = (\mathbf{t}_i, \mathbf{s}_i) / (\mathbf{t}_i, \mathbf{t}_i)$
  7. $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \hat{\mathbf{p}} + \omega_i \mathbf{z}_i$

8. *If* ($\mathbf{x}_{i+1}$ is accurate enough) STOP
   Else $\mathbf{r}_{i+1} = \mathbf{s}_i - \omega_i \mathbf{t}_i$

- END DO

3.2 Parallel Implementation of BICGSTAB Algorithm

With the chosen domain decomposition, each processor core takes care of the computation or updating of the vector fields such as **r**, **p**, **v**, **s**, **t** and **x** and matrix-vector products $\mathbf{A}\hat{\mathbf{p}}$ and $\mathbf{Az}$ for the blocks mapped to it. It also computes partial inner products required in Steps 1, 4 and 6. These partial inner products are summed up using global communications to form the full inner products. Inter-processor communications are also required for updating the vector fields in the overlap regions to facilitate local computation of matrix-vector products in Steps 4 and 6 of BICGSTAB algorithm. Efficient parallel implementation of the preconditioner (Steps 3 and 5) is crucial to the overall parallel efficiency of the BICGSTAB algorithm. This part of implementation depends on the choice of the preconditioner and is discussed separately in the next section.

4. Parallel Preconditioners for BICGSTAB

Convergence rate, robustness and computational efficiency of the parallel Poisson solver based on BICGSTAB is critically dependent on the choice of preconditioner used in Steps 3 and 5 of the algorithm. Since the parallel implementation of the BICGSTAB solver is based on the domain decomposition used in the Navier-Stokes solver, it is only natural to look for the preconditioners which can be implemented by using minimal additions to the existing data structure. Domain decomposition based additive methods (e.g. additive Schwarz method) or multiplicative solvers (for example, geometric multigrid) come across as the most suitable methods to employ as parallel preconditioners in our implementation. Another method which is inherently parallel and can be easily implemented in domain decomposition framework is Jacobi's method (also called the diagonal preconditioner). Although the convergence of BICGSTAB iterations would be very slow with Jacobi preconditioner, it provides a convenient benchmark for comparison of the convergence behaviour and performance of other preconditioners. We can also try a block SOR solver (with a fixed number of iterations) as a preconditioner which should provide an improvement over Jacobi preconditioner in terms of convergence of BICGSTAB iterations.

Let us note that the preconditioner steps return correction vectors required in BICGSTAB implementation. The theory of Krylov subspace methods requires that the preconditioner represents the same linear operator at each iteration. Jacobi's method or multigrid based preconditioners do satisfy this requirement. However, an additive Schwarz or SOR preconditioner based on an inexact subdomain solver would violate this requirement, making it inadmissible in theory. However, it has been observed that inexact solvers are acceptable in practice if the subdomain problems are solved fairly accurately [19,39]. In view of this observation, the preconditioner need not solve the linear system $\mathbf{M}\hat{\mathbf{p}} = \mathbf{p}_i$ (in Step 3) or $\mathbf{Mz}_i = \mathbf{s}_i$ (in Step 5) very accurately; approximate estimates of these vectors would be good enough for the BICGSTAB iterations. In subsequent discussions in this section, we shall presume that the preconditioner is required to return an approximate solution of the linear system of the form **Mz = s**.

4.1 Jacobi and SOR(k) Preconditioners

In Jacobi preconditioner, the preconditioner is diagonal part of **A**, i.e. **M** = diag(**A**). Thus, the effect of the preconditioner is given by

$$z_i = s_i / A_{ii} \tag{7}$$

All the data required in Eq. (7) for a subdomain is available on the processor assigned to it. Hence, parallel implementation of Jacobi preconditioner is straight-forward.

The SOR(k) preconditioner is formally represented as $\mathbf{M} = \left(diag\left(\mathbf{A}\right) - \omega\mathbf{E}\right)/\omega$ where matrix **E** is the strict lower triangular part of **A** and ω is over-relaxation parameter [9]. In domain decomposition framework, parallel SOR preconditioner can be implemented as a block SOR solver with a fixed number of iterations k in which each processor performs k iterations of the standard SOR method for the sub-problems corresponding to the blocks mapped on it. Thus, for each block, it computes the iterates as

$$z_i^{l+1} = \frac{\omega}{A_{ii}}\left(s_i - \sum_{j=1}^{i-1} A_{ij} z_j^{l+1} - \sum_{j=i+1}^{N} A_{ij} z_j^{l} \right) + (1-\omega)\, z_i^{l}, \quad l = 0,1,...,k \tag{8}$$

Note that in this implementation, new values of all iterates will not be available for the computations for boundary cells of a subdomain as those values are likely to be on different processors. Global communications must be performed after each iteration l to update the values of **z** for the cells in the guard-planes (i.e. the overlap region).

4.2 Additive Schwarz Preconditioner (ASM)

The additive Schwarz preconditioner is similar to the block Jacobi iteration, and essentially consists of the solution of sub-problems $\mathbf{A}^{\alpha}\mathbf{z}^{\alpha} = \mathbf{s}^{\alpha}$ corresponding to all the subdomains α in parallel. Effect of the preconditioner, **z,** is union of all the local vectors $\mathbf{z}^{\alpha}$ which are available as part of the block-data structure for each subdomain after solution of the local problems. No global communications are required as part of the additive Schwarz preconditioner, and hence, this preconditioner provides maximal parallelism.

Subdomain problems can be solved using any suitable direct or iterative method. In this paper, we have used SOR as the iterative solver for the subdomain problems. We have opted for a fixed number of SOR iterations for inexact subdomain solves.

4.3 Geometric Multigrid Preconditioner (GMG)

On Cartesian structured grids, the geometric multigrid method can be employed as solver as well as preconditioner to Krylov subspace methods. The preconditioner option is especially attractive for problems on complex domain in conjunction with the *fictitious domain* method. On uniform Cartesian grids, the fictitious domain multigrid preconditioner with conjugate gradient method has been shown to be a robust, efficient and scalable parallel Poisson solver [17]. Hence, it is worthwhile to design and explore the effectiveness of a parallel fictitious domain multigrid preconditioner for non-uniform grids. For full algorithmic details of this fictitious domain preconditioner, see Singh and Willams [17]. For sake of brevity, we would refer this fictitious domain geometric multigrid preconditioner as GMG (or simply, multigrid) preconditioner in this paper.

On non-uniform grids, there are two options for implementation of the multigrid scheme as a preconditioner. The first option is mapping from a non-uniform grid to an auxiliary uniform Cartesian grid on which the multigrid method can be easily implemented. The second approach is the generation of nested grids and relevant operators on the given non-uniform grid. Regarding the first approach, Douglas et al. [40] suggest that although it is good for moderately graded unstructured grids, it should be avoided for stretched structured grids in CFD simulations since the mapping can lead to a system with a high condition number which eliminates advantages of multigrid. Hence, we focus on the second approach in this work.

Note that the generation of a hierarchy of grids required in the multigrid method is very simple on structured Cartesian grids (whether uniform or non-uniform). This process is further simplified if the number of grid divisions in each direction are chosen as some power of two. The main computational challenge lies in the generation of multi-grid operators. There are two options for generation of system matrices on each grid of the multigrid hierarchy: (a) use of Galerkin approach and (b) use of the chosen discretization process on each grid. The former approach is conceptually more elegant in the sense that it also provides the inter-grid transfer operators. However, this purely algebraic process is computationally intensive, and would also require

substantial additional memory for storage of these operators. The second approach provides a computationally efficient route for generation of system matrices on each grid. However, generation of restriction and prolongation operators is still based on the Galerkin approach. We have opted for a *V*-cycle multigrid. To summarize, our geometric multigrid preconditioner consists of the following components:

- Grid hierarchy based on simple agglomeration of two adjacent cells at the finer grid level.
- Generation of system matrices using the cell-centred finite volume / finite difference discretization (i.e. using the same discretization scheme which was used at the finest grid).
- Generation of restriction and prolongation based on the Galerkin approach.
- Red-black Gauss-Seidel iterations as smoothing/relaxation procedure.

5. Numerical Results

For evaluation of the performance of the parallel preconditioners for solution of the pressure Poisson equation, we consider an LES of a marine turbine [38]. In this paper, we confine ourselves to the aspects relevant to the solution of the pressure Poisson equation only. Geometry of the turbine is shown in Figure 1. The non-uniform mesh used for the simulations is based on stretched Cartesian grid in which a uniform fine mesh has been used in a region around the rotor, and a progressively stretched grid has been used away from this core region. The base mesh used for LES of the marine turbine [38] has a stretch factor of 1.05 with a maximum aspect ratio limit of 10. Figure 2 gives a close-up of the mesh around the rotor. For this study of pressure Poisson solvers, we have used a relatively coarse global mesh of 512×384×384 in *X*, *Y*, *Z* directions respectively. Numerical simulations have been performed using an in-house explicit Navier-Stokes code CgLES based on immersed boundary method. For parallel simulations, the computational domain has been decomposed in 288 sub-domains (8×6×6 blocks along *X*, *Y*, *Z* directions). Each block contains 64×64×64 grid. For full details of the computational domain and methodology used for large eddy simulation, see Bai et al. [38].

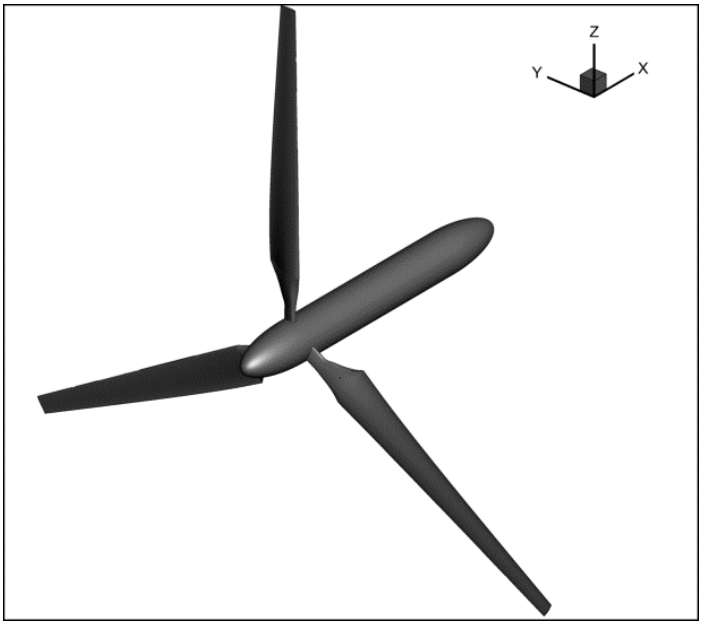


Figure 1. Geometry of marine current turbine

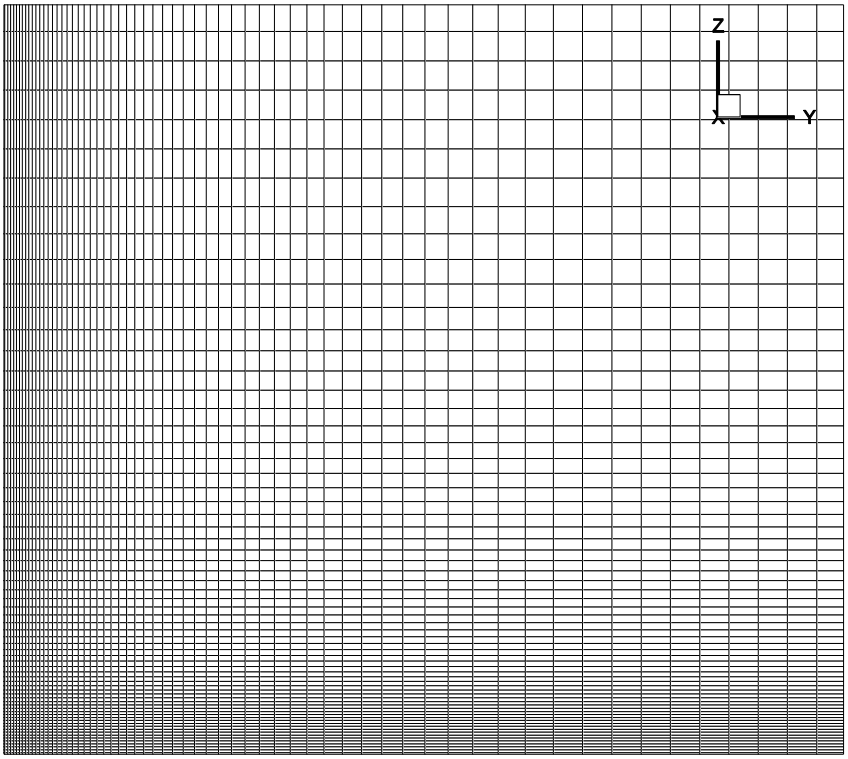


Figure 2. Close-up of view of mesh slice showing grid stretching (stretch factor = 1.05)

To study the effect of the grid stretching, we have performed numerical simulations with two other grids with a stretching factor of 1.10 and 1.20 respectively. Thus, the three grids used in this study are

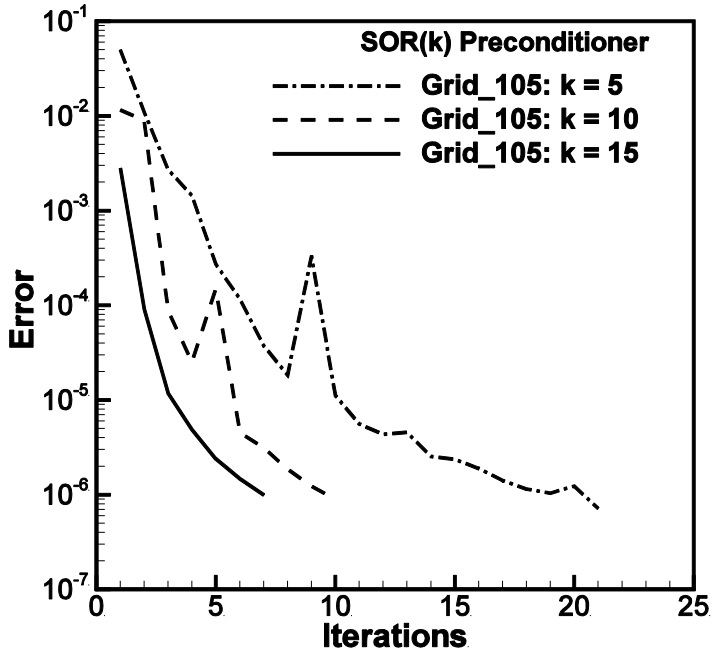
- **Grid_105**: Stretch factor of **1.05** and maximum permissible aspect ratio of 10
- **Grid_110**: Stretch factor of **1.10** and maximum permissible aspect ratio of 20
- **Grid_120**: Stretch factor of **1.20** and maximum permissible aspect ratio of 20

Let us note that the last two grids have not been used in the actual LES simulations of the marine turbine. These have been artificially created for studying the effect of the grid stretching on the performance of different preconditioners for solving the pressure Poisson equation. All computations have been performed using 32 cores on the joint Minerva HPC Cluster of Queen Mary University of London and the University of Warwick, UK. A relative tolerance of 1.0E-06 has been used as the convergence criterion for the BICGSTAB iterations.

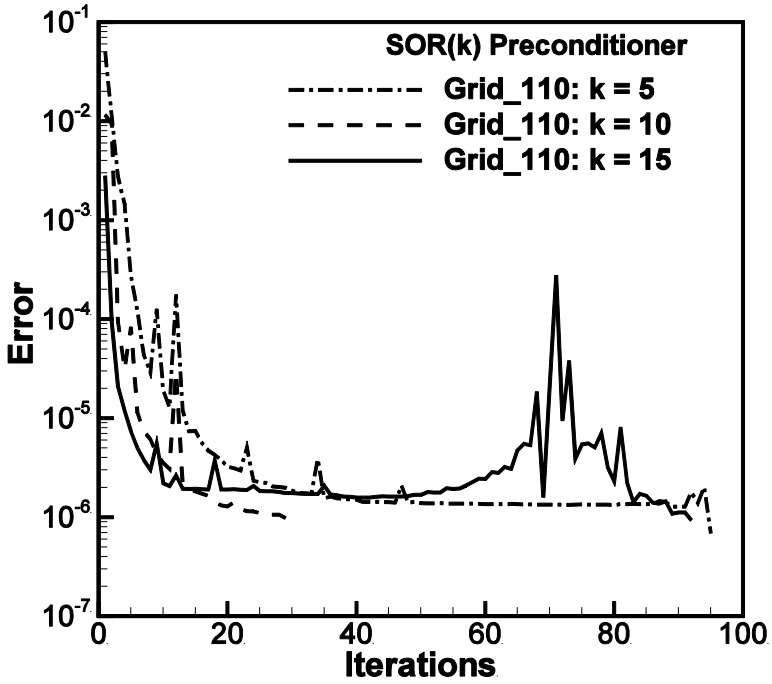
5.1 Performance of SOR(k) Preconditioner

To assess the performance of the SOR(*k*) preconditioner, computations have been performed with different values of *k* for all the grids. Table 1 and Figure 3 summarize results of the convergence properties and computational efficiency of the BICGSTAB solver for a representative time step. With increase in *k*, we expect better convergence of the BICGSTAB iterations. With all three grids, rapid rate convergence is observed in the initial stages of the BICGSTAB iterations for all three values of *k*. However, the rate of convergence diminishes rapidly thereafter, and shows markedly non-uniform convergence behaviour. For the base LES grid (Grid_105) with a moderate grid stretching, convergence behaviour improves with increase in value of *k*. However, similar pattern is not observed for highly stretched grids (Grid_110 and Grid_120). On an overall, the choice of k = 10 emerges as the optimum choice from view point of convergence behaviour as well the computational efficiency as evidenced from the CPU time estimates in Table 1.

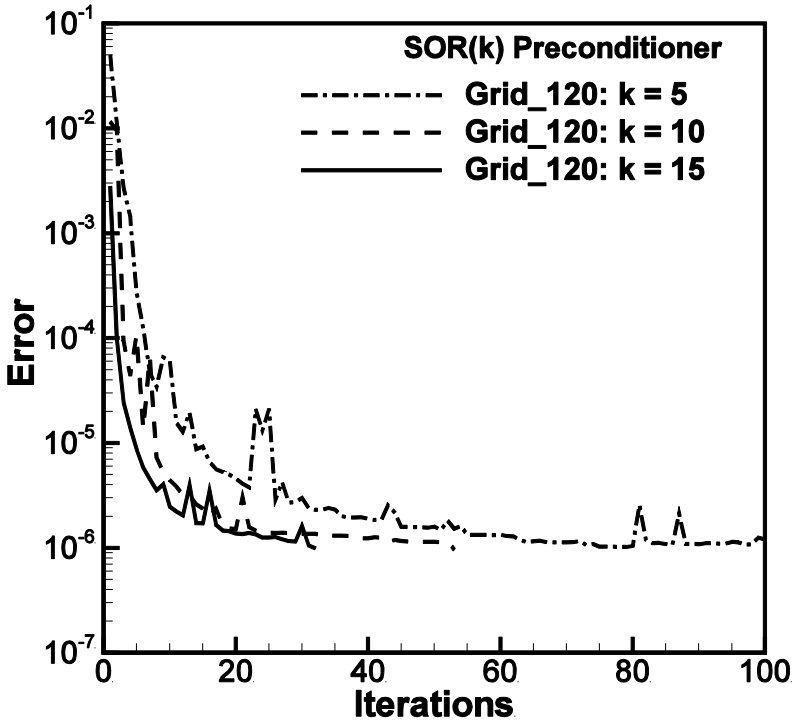
Table 1. SOR(*k*) preconditioner: Effect of number of SOR iterations, ***k*** on convergence of BiCGSTAB solver. (Time indicates wall clock time in seconds)									
Grid	***k* = 5**			***k* = 10**			***k* = 15**		
	Iterations	Error	Time	Iterations	Error	Time	Iterations	Error	Time
Grid_105	21	7.2×10^{-7}	28.35	10	8.9×10^{-7}	23.72	7	9.9×10^{-7}	23.73
Grid_110	95	6.9×10^{-7}	129.3	29	9.8×10^{-7}	68.97	92	9.4×10^{-7}	314.0
Grid_120	155	9.6×10^{-7}	215.0	53	9.6×10^{-7}	126.2	32	9.9×10^{-7}	108.6



(a) Convergence of SOR(k) preconditioner for Grid_105



(b) Convergence of SOR(k) preconditioner for Grid_110



(c) Convergence of SOR(k) preconditioner for Grid_120

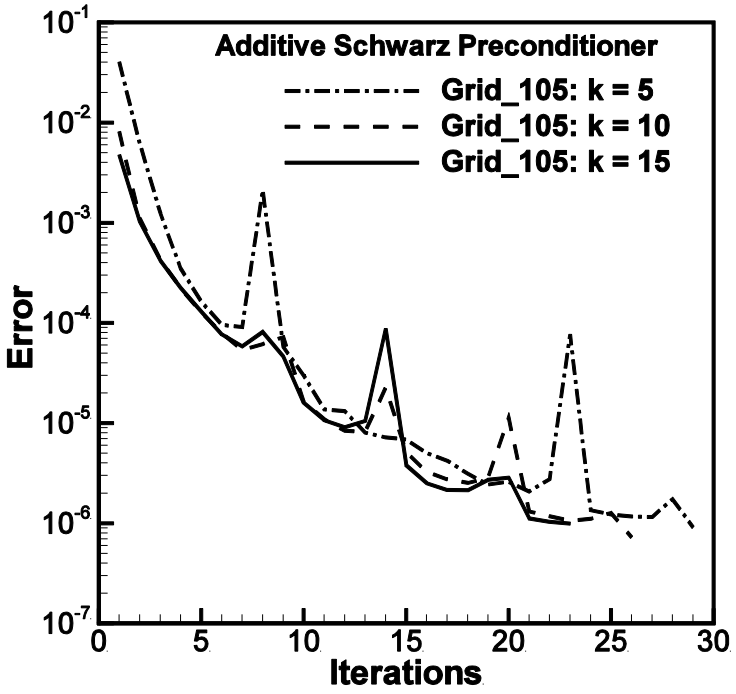
Figure 3: Convergence of BICGSTAB iterations with SOR(k) preconditioner

5.2 Performance of Additive Schwarz (ASM) Preconditioner

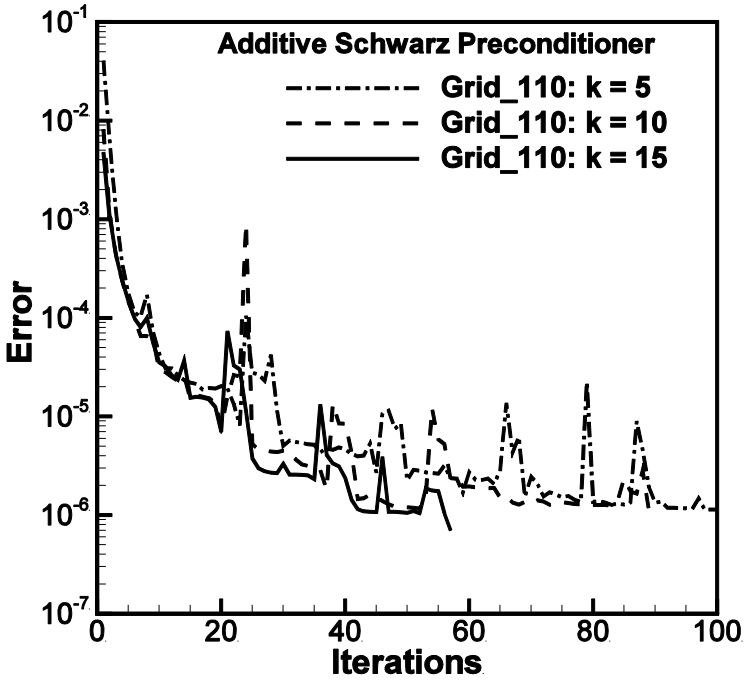
For the additive Schwarz preconditioner, SOR has been used as the sub-domain solver. Computations have been performed with different values of *k* as defined in Eq. (8) and results obtained for three grids are summarized in Table 2 and Figure 4 for a representative time step. Again, rapid rate convergence is observed in the initial stages of BICGSTAB iterations for all three values of *k*, whereas erratic convergence pattern is observed in the later stages of BICGSTAB iterations. For moderately stretched grids (Grid_105 and Grid_110), the choice of $k = 10$ for subdomain solves shows a fairly smooth convergence behaviour, and thus should be preferred choice. At the same time, there is no clear-cut choice of *k* with additive Schwarz preconditioner for all grids (as we have observed with SOR(*k*) preconditioner).

Table 2. Additive-Schwarz preconditioner: Effect of number of SOR iterations, ***k*** on convergence of BiCGSTAB solver. (Time indicates wall clock time in seconds)

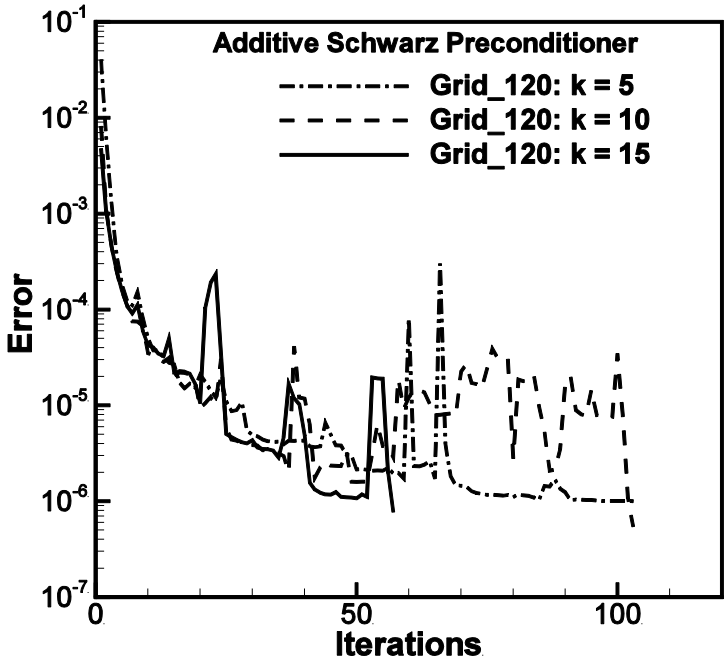
Grid	***k* = 5**			***k* = 10**			***k* = 15**		
	Iterations	Error	Time	Iterations	Error	Time	Iterations	Error	Time
Grid_105	29	9.2×10^{-7}	36.11	26	7.2×10^{-7}	55.25	23	9.9×10^{-7}	69.13
Grid_110	110	7.3×10^{-7}	138.5	90	9.1×10^{-7}	191.3	57	7.0×10^{-7}	172.7
Grid_120	218	7.8×10^{-7}	274.9	103	9.8×10^{-7}	218.8	57	3.4×10^{-7}	173.4



(a) Convergence of additive Schwarz preconditioner for Grid_105



(b) Convergence of additive Schwarz preconditioner for Grid_110



(c) Convergence of additive Schwarz preconditioner for Grid_120

Figure 4: Convergence of BICGSTAB iterations with additive Schwarz preconditioner (with varying number of SOR iterations *k* used in sub-domain solves)

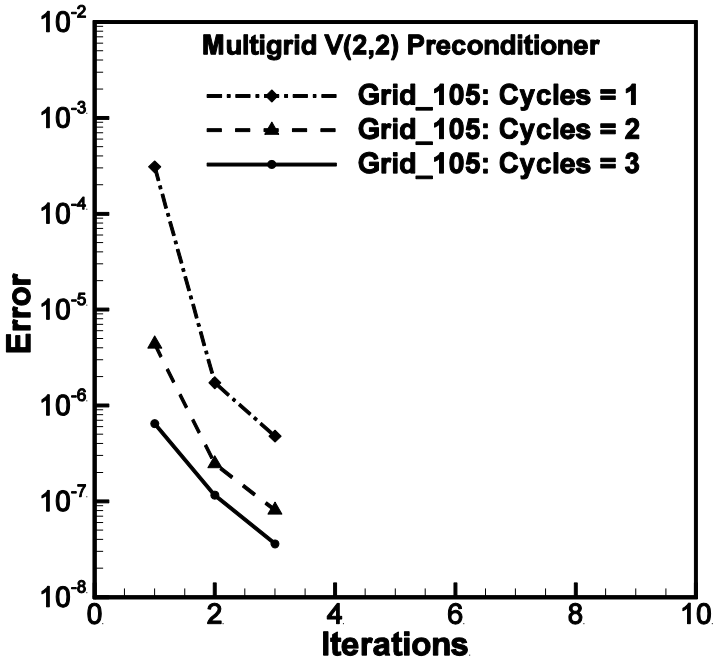
5.3 Performance of Geometric Multigrid (GMG) Preconditioner

When multigrid is used as a preconditioner for Krylov subspace solvers, one would normally use only a few multigrid cycles to obtain the effect of the preconditioners. Further, a small number of pre- and post-smoothing (one or two) iterations have normally been used in the literature. Thus, we need to explore the effect of two set of parameters on performance of the fictitious domain multigrid preconditioner: (a) number on multigrid cycles, and (b) number of smoothing iterations. We would use the notation $V(p,q)$ to represent a V-cycle multigrid with p pre-smoothing and q post-smoothing iterations.

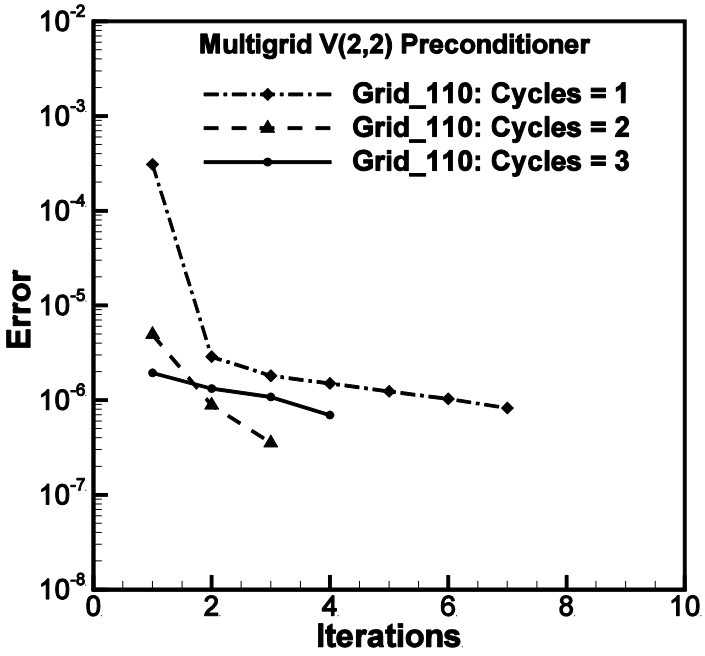
Let us first explore the effect of number of multigrid cycles on convergence of BICGSTAB iterations with the $V(2,2)$ multigrid cycle. Results of computations for different grids are summarized in Table 3 and Figure 5. For the base LES grid (Grid_105) with a moderate grid stretching, best convergence behaviour and computational efficiency is observed with use of three multigrid cycles. However, similar pattern is not observed for more stretched grids (Grid_110 and Grid_120) with this choice. For moderately stretched grids (Grid_105 and Grid_110), use of two multigrid cycles gives consistently good performance (in terms of convergence as well as computing time). Further, although the choice of only one multigrid cycle may not have the best computational efficiency on all grids, it shows consistent convergence pattern in all the cases. Thus, it would be advisable to stick to the use of one or at most two multigrid cycles in the multigrid preconditioner.

Table 3. Effect of number of *V*(2,2) multigrid cycles on performance of multigrid preconditioner (Time indicates wall clock time in seconds)

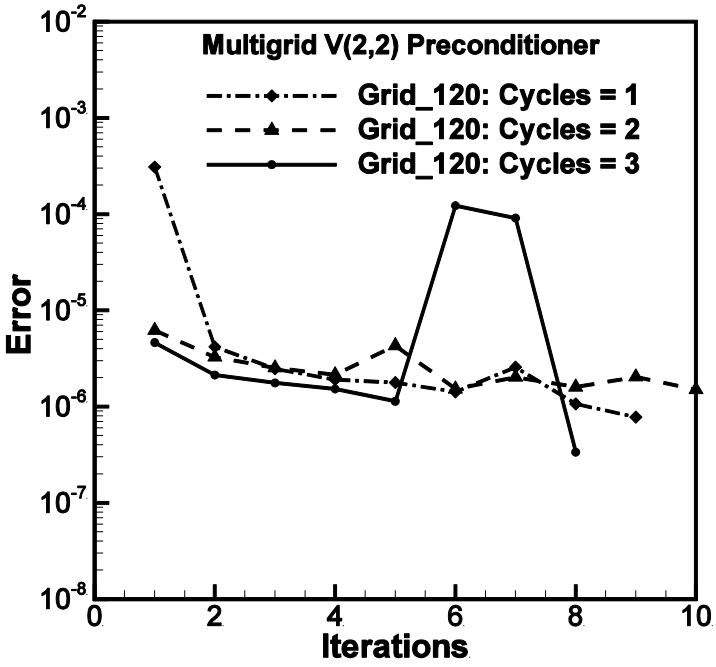
Grid	MG Cycles = 1			MG Cycles = 2			MG Cycles = 3		
	Iterations	Error	Time	Iterations	Error	Time	Iterations	Error	Time
Grid_105	3	4.8×10^{-7}	4.85	2	2.4×10^{-7}	5.82	1	6.5×10^{-7}	4.15
Grid_110	7	8.3×10^{-7}	11.45	2	8.8×10^{-7}	5.77	4	6.9×10^{-7}	16.49
Grid_120	9	7.8×10^{-7}	14.70	22	9.8×10^{-7}	62.94	8	3.4×10^{-7}	33.16



(a) Convergence of multigrid preconditioner for Grid_105



(b) Convergence of multigrid preconditioner for Grid_110



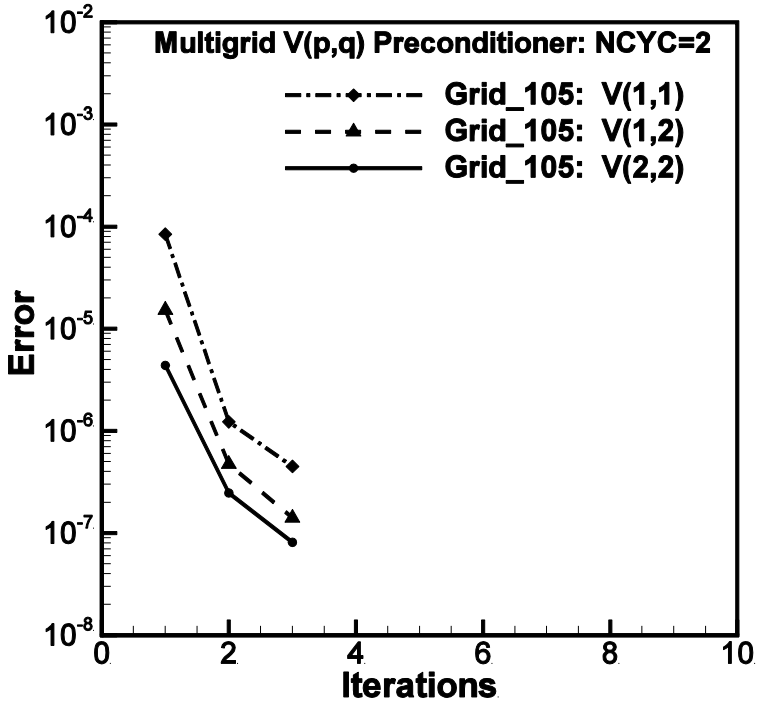
(c) Convergence of multigrid preconditioner for Grid_120

Figure 5: Convergence of BICGSTAB iterations with fictitious domain **multigrid** preconditioner with different number of multigrid cycles

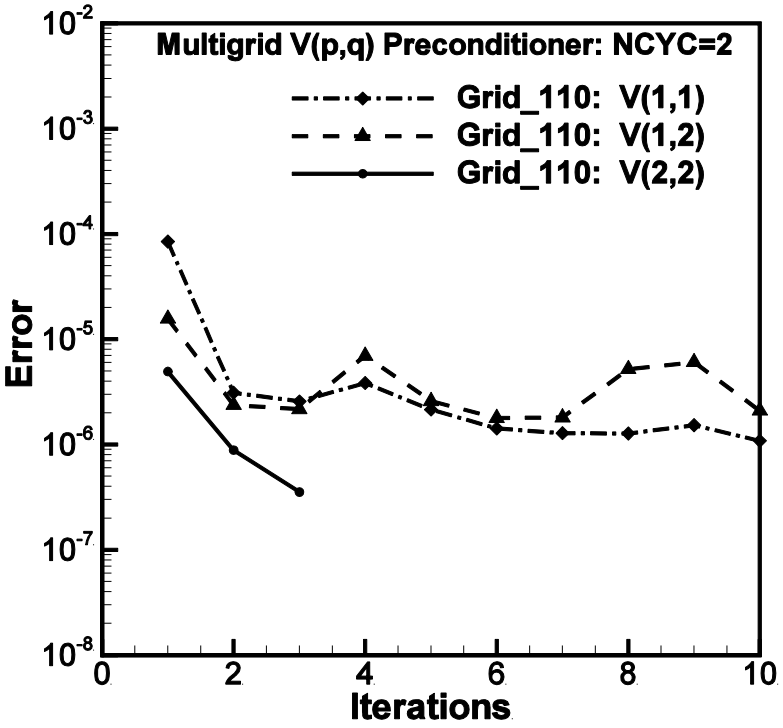
Next, let us explore the effect of varying number of pre- and post-smoothing iterations, i.e. the use of different $V(p,q)$ cycles. Results obtained with three multigrid cycles with different choices of p and q are summarized in Table 4 and Figure 6. Best performance can be observed with $V(2,2)$ cycle, especially for moderately stretched grids (Grid_105 and Grid 110). Thus, $V(2,2)$ multigrid cycle should be preferred over $V(1,1)$ and $V(1,2)$ cycles in construction of the fictitious domain multigrid preconditioner for BICGSTAB iterations.

Table 4. Effect of number of pre- and post-smoothing iterations on performance of multigrid preconditioner: $V(p,q)$ represents V-cycle with p pre-smoothing and q post-smoothing iterations (Number of multigrid cycles = 2) (Time indicates wall clock time in seconds)

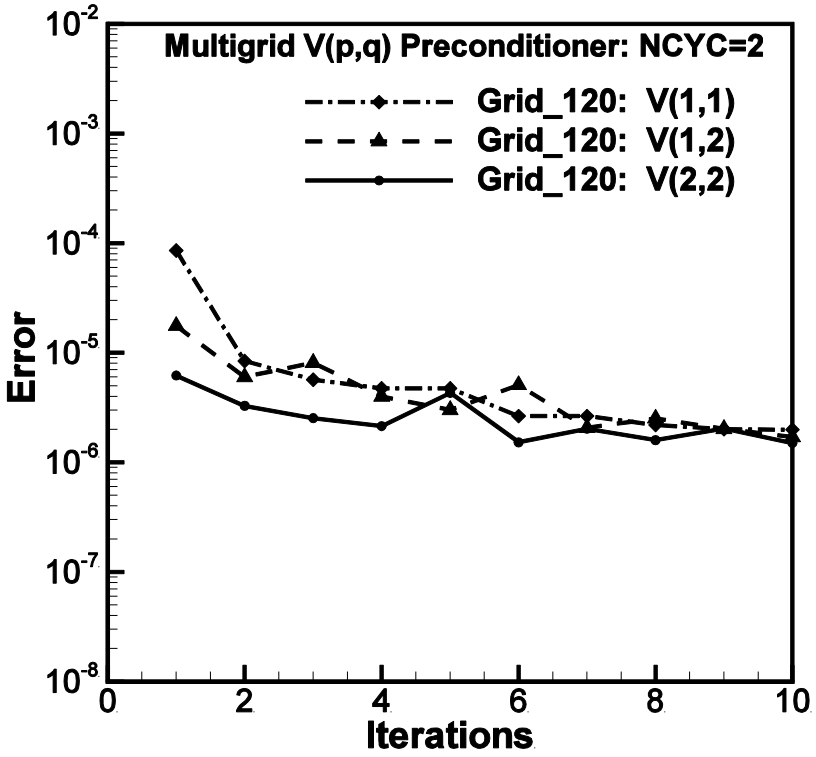
Grid	***V*(1,1)**			***V*(1,2)**			***V*(2,2)**		
	Iterations	Error	Time	Iterations	Error	Time	Iterations	Error	Time
Grid_105	3	4.5×10^{-7}	6.58	2	4.7×10^{-7}	5.10	2	2.4×10^{-7}	5.82
Grid_110	14	9.5×10^{-7}	30.90	13	7.8×10^{-7}	33.29	2	8.8×10^{-7}	5.77
Grid_120	19	9.9×10^{-7}	41.80	17	6.9×10^{-7}	43.44	22	9.8×10^{-7}	62.94



(a) Convergence of multigrid preconditioner for Grid_105



(b) Convergence of multigrid preconditioner for Grid_110



(c) Convergence of multigrid preconditioner for Grid_120

Figure 6: Convergence of BICGSTAB iterations with fictitious domain **multigrid** preconditioner with different multigrid $V(p,q)$ cycles

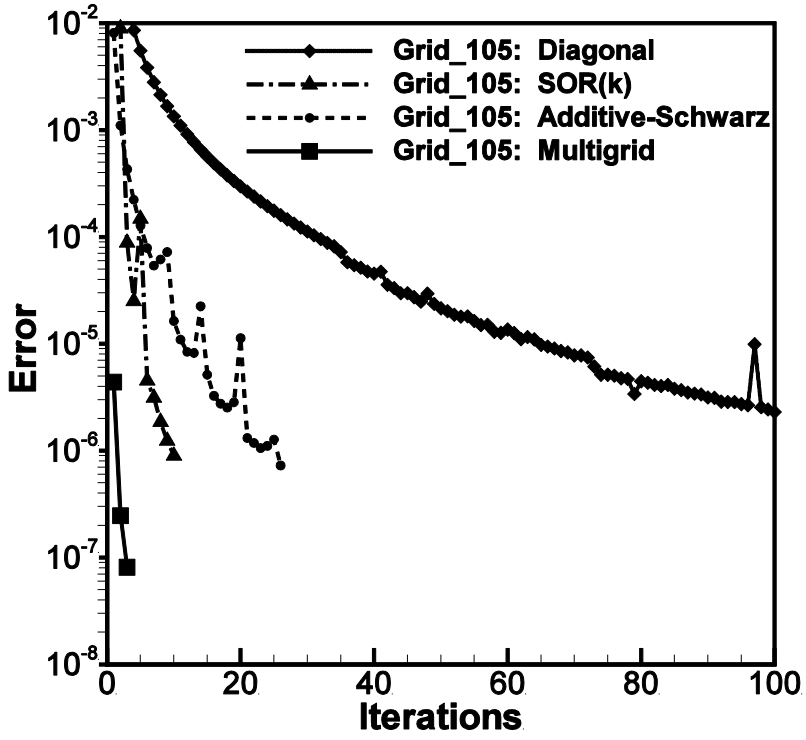
5.4 Comparison of Performance of Different Preconditioners

Convergence plots and timing estimates in the previous sections provide us an idea of the relative performance of different preconditioners for BICGSTAB iterations. A clearer picture of the comparative performance of different preconditioners is provided by iteration and timing estimates in Table 5 and convergence plots in Figure 7. For sake of comparison, we have also included results with Jacobi (diagonal) preconditioner. We can clearly see that the multigrid preconditioner shows the best convergence behaviour followed by the SOR(k) preconditioner. Similar pattern holds with respect to computing time: the multigrid preconditioner requires least computing time for a given convergence tolerance. The next best preconditioner is SOR(k) with k = 10.

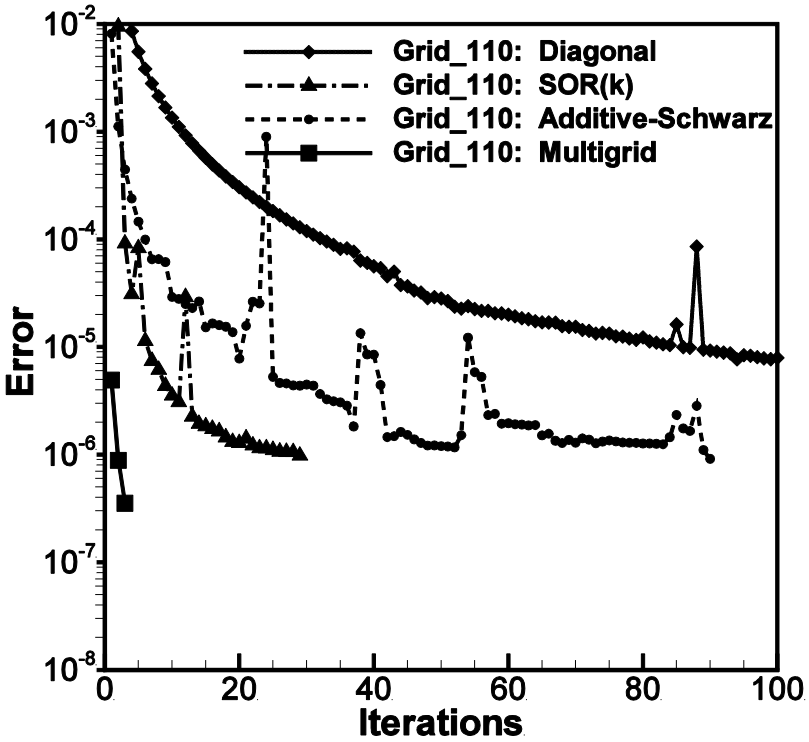
Table 5. Iteration counts and computing time estimates for BICGSTAB solver with different preconditioners. Time indicates wall clock time in seconds. Additive-Schwarz preconditioner is based on 10 SOR iterations for subdomain solves; Multigrid preconditioner consists of two V(2,2) cycles.

Grid	***Jacobi***		***SOR(k)* (*k* = 10)**		**Additive-Schwarz**		**Multigrid**	
	Iterations	Time	Iterations	Time	Iterations	Time	Iterations	Time
Grid_105	140	66.73	10	23.72	26	55.25	2	5.82
Grid_110	415	198.3	29	68.97	90	191.3	2	5.77
Grid_120	----*	-----*	53	126.2	103	218.8	22	62.94

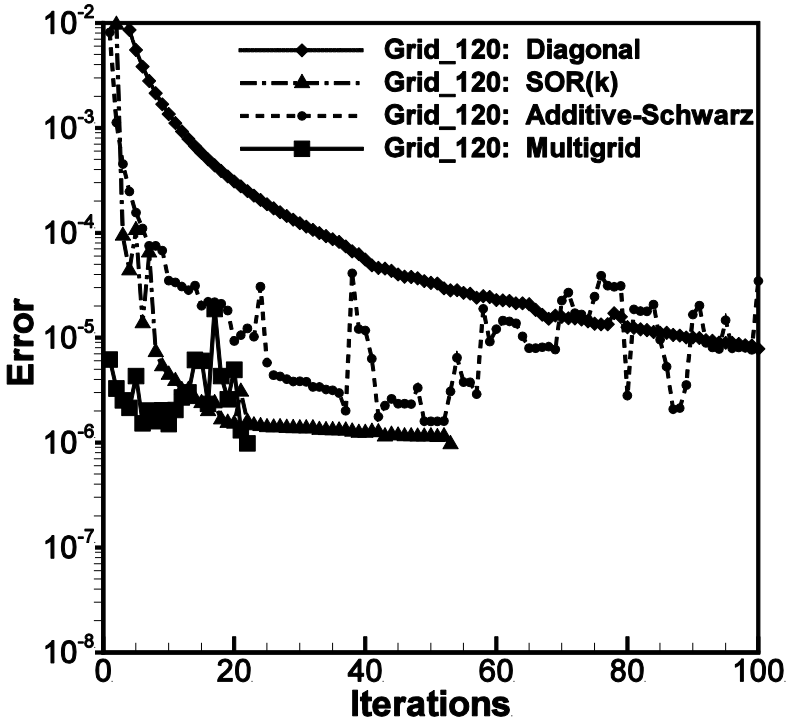
* Did not converge to specified tolerance of 1.0E-06 in 10000 iterations



(a) Convergence of various preconditioners for Grid_105



(b) Convergence of various preconditioners for Grid_110



(c) Convergence of various preconditioners for Grid_120

Figure 7: Convergence of BICGSTAB iterations with different preconditioners. SOR(k) results correspond to k = 10, additive-Schwarz results are based on 10 SOR iterations for sub-domain solves, and multigrid results are obtained using two V(2,2) cycles.

5.5 Effect of Grid Refinement on Performance of Preconditioners

Preceding results provide an estimate of performance of different preconditioners for a single grid (with different stretch factors). To get a more complete picture of the performance of these preconditioners, we have carried out simulations on a fine mesh of 2X resolution as compared to the coarse grid employed for preceding simulations. The fine mesh consists of 1024×768×768 grid points in *X*, *Y*, *Z* directions respectively (approximately 600 million cells). For parallel simulations, the computational domain has again been decomposed in 288 sub-domains (8×6×6 blocks along *X*, *Y*, *Z* directions). Each subdomain contains a 128×128×128 grid.

Performance of SOR(*k*) and Additive-Schwarz Preconditioners

Fine grid results with the SOR(*k*) and additive Schwarz preconditioners are summarized in Table 6 and Figures 8 and 9 for a representative time step. Both of these preconditioners exhibit fairly similar convergence behaviour for the fine grid with all the three grid stretching. For the base LES grid (Grid_105) with a moderate grid stretching, very smooth and rapid convergence is observed with SOR preconditioner for both values of k (k = 10 and 15). However, choice of k = 15 gives better convergence for all three stretched grids. SOR(k) preconditioner can be observed to be more efficient than additive Schwarz preconditioner from the CPU time estimates in Table 6. This trend is very similar to that observed with coarse grid simulations in preceding sections.

Table 6 . Performance of SOR(*k*) and additive-Schwarz preconditioners for the fine grid. (Time: wall clock time in seconds)

Grid	SOR*(k)*						Additive-Schwarz		
	***k* = 10**			***k* = 15**			**(*k* = 15)**		
	Iterations	Error	Time	Iterations	Error	Time	Iterations	Error	Time
Grid_105	41	8.55×10^{-7}	204	30	7.42×10^{-7}	215	72	8.13×10^{-7}	476
Grid_110	718	9.88×10^{-7}	3546	429	9.52×10^{-7}	3070	796	8.99×10^{-7}	5253
Grid_120	* ---	* ---	* ---	525	3.87×10^{-7}	3721	844	8.91×10^{-7}	5672

* Did not converge to specified tolerance of 1.0×10^{-6} in 1000 iterations.

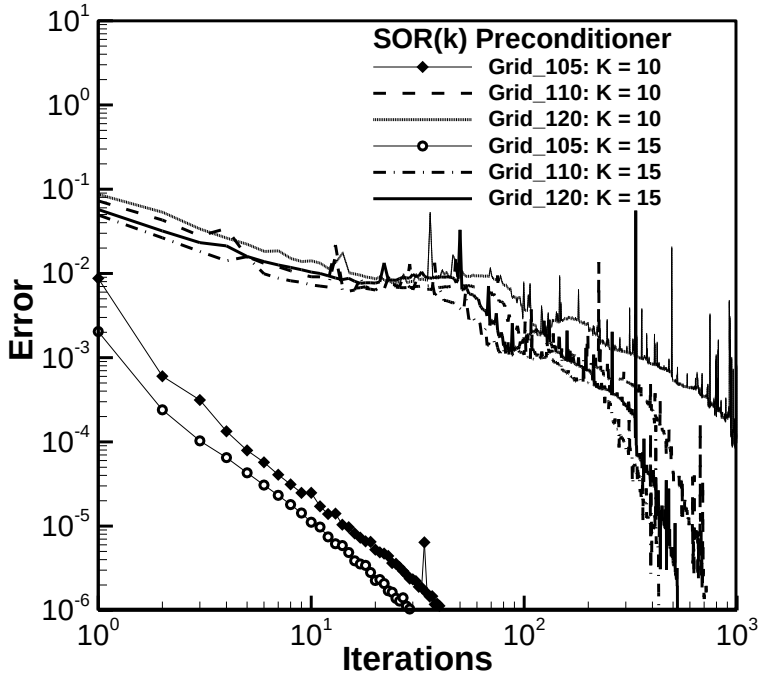


Figure 8: Convergence of BICGSTAB iterations with SOR(k) preconditioner for the fine grid

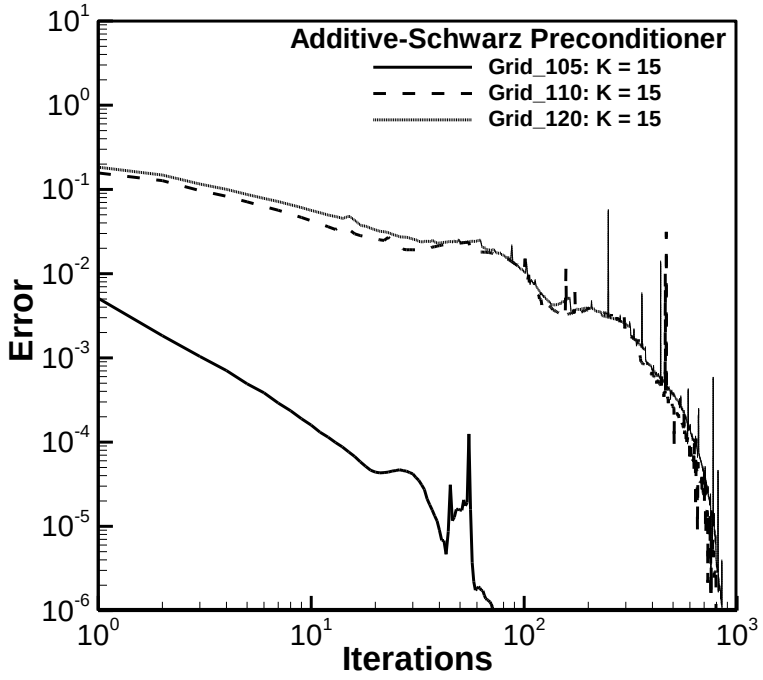


Figure 9: Convergence of BICGSTAB iterations with additive Schwarz preconditioner for the fine grid

Performance of Multigrid Preconditioner

Convergence history for the fine grid simulations with multigrid preconditioner is summarized in Figure 10 for the selected grid stretch factors. Effect of the grid anisotropy on convergence is again very similar to that observed in coarse grid simulations. The multigrid preconditioner shows excellent convergence behaviour for the moderately stretched grids (Grid_105 and Grid_110). Convergence of multigrid preconditioner is not very smooth for the excessively stretched grid (Grid_120), but is still much better than that observed with SOR(*k*) (Figure 8) or additive Schwarz preconditioners (Figures 9). Further, to observe the effect of the number of degrees of freedom on convergence behaviour, we have plotted the convergence history of BiCGSTAB iterations with multigrid and SOR(*k*) preconditioners for the coarse as well as find grid in Figure 11. It can be clearly seen that the convergence behaviour of BiCGSTAB iterations is more a function of grid anisotropy (stretching) than the number of degrees of freedom for both the preconditioners. For the base LES grid (Grid_105), convergence of the multigrid preconditioner is seen to be almost independent of the number of degrees of freedom.

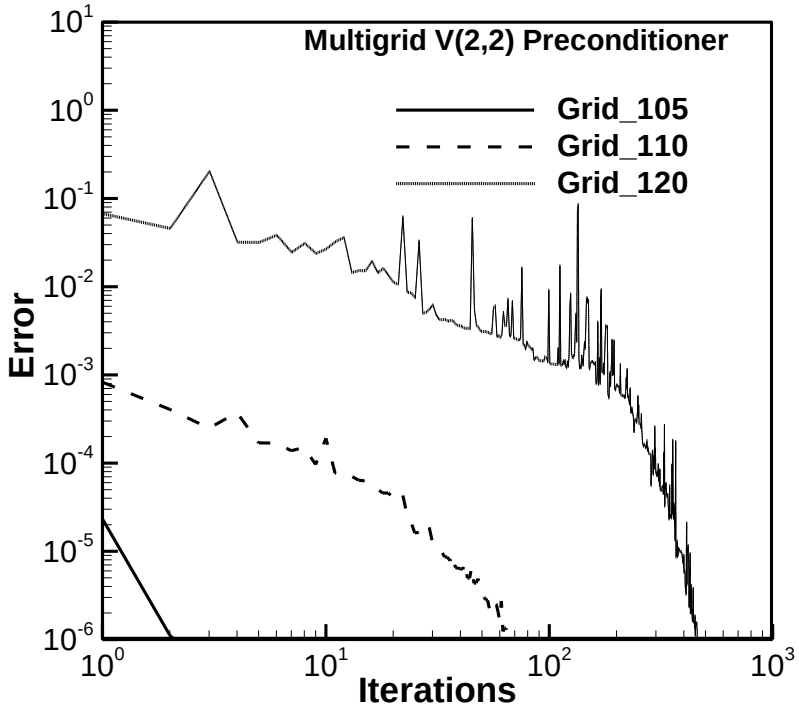
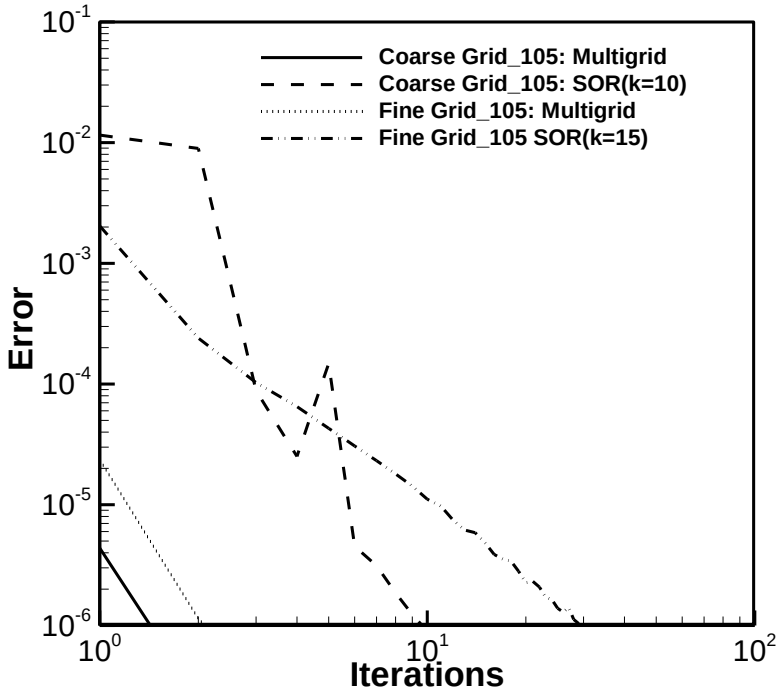
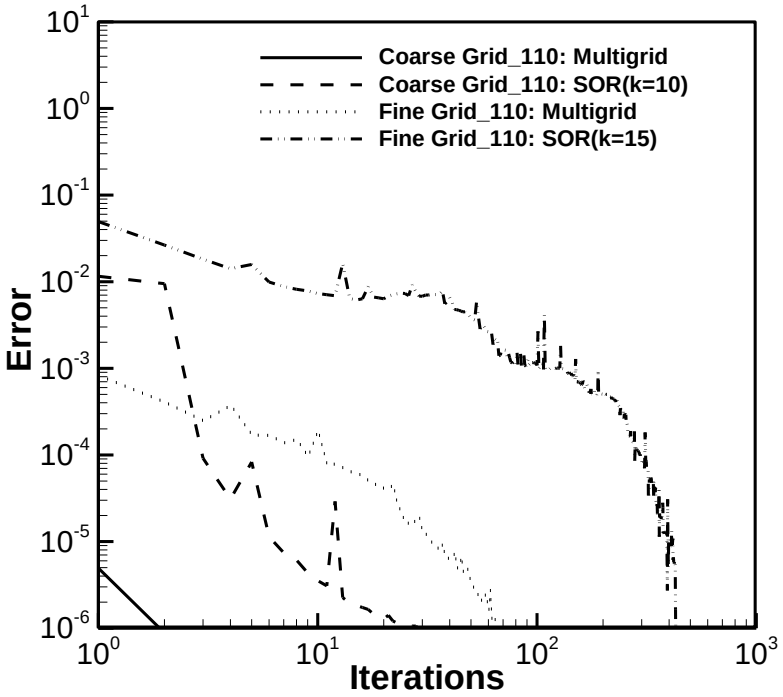


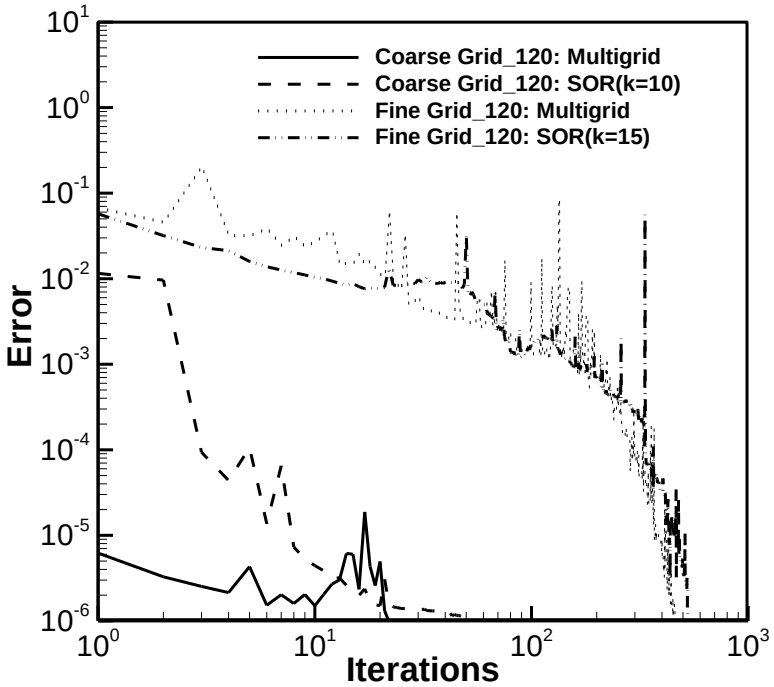
Figure 10: Convergence of BICGSTAB iterations with multigrid preconditioner for fine grid



(a) Convergence of multigrid and SOR(k) preconditioners for Grid_105



(b) Convergence of multigrid and SOR(k) preconditioners for Grid_110



(c) Convergence of multigrid and SOR(k) preconditioners for Grid_120

Figure 11: Convergence of BICGSTAB iterations with multigrid and SOR(k) preconditioners for different grid stretch factors for coarse as well as fine grid.

Table 7 presents a comparison of computational efficiency (in terms of CPU time estimates) for fine grid simulations with multigrid, SOR(*k*) and additive Schwarz preconditioners. Once again, the multigrid preconditioner emerges as the most efficient choice: it is more efficient by an order of magnitude for the moderately stretched grids (Grid_105 and Grid 110) which are commonly

employed in LES/DNS of turbulent flows. For highly stretched grid (Grid_120), SOR(k) can be seen to be its close competitor.

Table 7. Performance of multigrid preconditioner (V(2,2) cycle with number of multigrid cycles = 2) as compared to SOR and additive Schwarz preconditioners for the fine grid. (Time indicates wall clock time in seconds).

Grid	Multigrid		SOR(k), k = 15		Additive-Schwarz (k=15)	
	Iterations	Time	Iterations	Time	Iterations	Time
Grid_105	3	19	30	215	72	476
Grid_110	65	405	429	3070	796	5253
Grid_120	458	2844	525	3721	844	5672

Preceding numerical results for a coarse grid as well as a fine grid clearly establish the superior performance of the geometric multigrid preconditioner for stretched grids involving mild to moderate stretching. For these grids, multigrid preconditioner requires CPU time an order of magnitude lower than SOR(k) and additive Schwarz preconditioners. However, performance of the multigrid preconditioner is not as good for highly stretched grids. This behaviour could be attributed to the insufficient smoothing provide by red-black Gauss-Seidel smoother used in present implementation. Therefore, to improve the performance of multigrid preconditioner for these highly anisotropic grids, more effective smoothers of ILU type would be required. Nevertheless, even for the grids involving excessive stretching (stretch factor of 1.2), multigrid preconditioner gives better performance than the other two preconditioners. Further, the SOR(*k*) preconditioner is seen as the next best alternative, especially for highly stretched grids.

Thus, we have successfully implemented and tested an efficient pressure Poisson solver for immersed boundary Navier-Stokes solver on non-uniform grids for LES/DNS of turbulent flow in complex geometries. This Poisson solver is based on the BiCGSTAB method with a set of parallel preconditioners for arbitrarily graded Cartesian grids. The set of preconditioners includes a geometric multigrid preconditioner, SOR(*k*) preconditioner and additive Schwarz preconditioner for stretched grids. Multigrid preconditioner emerges as the best choice followed by SOR(*k*) preconditioner.

6. Concluding Remarks

We have presented an assessment of a set of parallel pre-conditioners for numerical solution of the pressure Poisson equation arising in large eddy simulation of turbulent incompressible flows on non-uniform Cartesian grids. Since parallel implementation of our explicit Navier-Stokes solver is based on domain decomposition, we have considered pre-conditioners suitable for domain decomposition based parallel implementation of the pressure Poisson solver on non-uniform Cartesian grids. Bi-conjugate gradient stabilized (BICGSTAB) method has been adopted as the Krylov solver for the linear algebraic system resulting from the discretization of the Poisson equation. Numerical experiments have been performed to assess the performance of different parallel preconditioners such as Jacobi, SOR(*k*), additive Schwarz and multigrid preconditioners for difference mesh stretching. Numerical results clearly show the effectiveness and superior performance of multigrid preconditioner as compared to the Jacobi, SOR(*k*) or additive Schwarz preconditioners. The SOR(*k*) preconditioner emerges as the next best alternative.

Acknowledgements

This work was supported by Research Exchanges with China & India Award of Royal Academy of Engineering, UK to Drs Eldad Avital and Krishna M. Singh (Grant No. SEMF1A4R). Parallel computing facilities were extended by UK-EPSRC Turbulence Consortium (Grant No.

EP/L000261/1), Queen Mary University of London and the Centre of Scientific Computing, Warwick University, UK. Support of these organizations is gratefully acknowledged. The authors also gratefully acknowledge the suggestion for improvement received from the anonymous referees.

References

[1] J.H. Ferziger, M. Peric, Computational Methods for Fluid Dynamics, Springer-Verlag, Berlin (2003).

[2] G.H. Golub, L.C. Huang, H. Simon, W.-P. Tang, A Fast Poisson Solver for the Finite Difference Solution of the Incompressible Navier--Stokes Equations, SIAM J. Sci. Comput. 19 (1998) 1606–1624.

[3] R. Löhner, F. Mut, J.R. Cebral, R. Aubry, G. Houzeaux, Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation : Extensions and improvements, Int. J. Numer. Meth. Engng. (2011) 2–14.

[4] A. Segal, M. ur Rehman, C. Vuik, Preconditioners for Incompressible Navier-Stokes Solvers, Numer. Math. Theory Meth. Appl. 3 (2010) 245–275.

[5] H.-W. Hsu, F.-N. Hwang, Z.-H. Wei, S.-H. Lai, C.-A. Lin, A parallel multilevel preconditioned iterative pressure Poisson solver for the large-eddy simulation of turbulent flow inside a duct, Comput. Fluids. 45 (2011) 138–146.

[6] A. Mcadams, E. Sifakis, J. Teran, A parallel multigrid Poisson solver for fluids simulation on large grids, in: M. Otaduy and Z. Popovic (Editors), Proc. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2010.

[7] N. Zhao, X. Wang, A Parallel Preconditioned Bi-Conjugate Gradient Stabilized Solver for the Poisson Problem, J. Comput. 7 (2012) 3088–3095.

[8] E.J. Avital, A second look at the role of the fast Fourier transform as an elliptic solver, Int. J. Numer. Meth. Fluids, 48 (2005) 909–927.

[9] Y. Saad, Iterative Methods for Sparse Linear Systems, Second Edition, SIAM, 2003.

[10] U. Trottenberg, C.W. Oosterlee, A. Schüller, Multigrid, Academic Press, London, 2000.

[11] P. Wesseling, An Introduction to Multigrid Methods, R. T. Edwards, 2004.

[12] A. Brandt, N. Dinar, Multigrid solutions to elliptic flow problems, in: S. Parter (Ed.), Numerical Methods for Partial Differential Equations, Academic Press, New York, 1979: pp. 53–147.

[13] J.H. Bramble, Multigrid Methods, Longman Scientific & Technical, Essex, England, 1993.

[14] C.C. Douglas, Multigrid methods in science and engineering, IEEE Comput. Sci. Eng. 3 (1997) 55–68.

[15] W. Hackbusch, U. Trottenberg, Multigrid Methods, Springer-Verlag, Berlin, 1982.

[16] J.W. Ruge, K. Stüben, Algebraic multigrid (AMG), in: S.F. McCormick (Ed.), Multigrid Methods, SIAM, Philadelphia, PA, 1987: pp. 73–130.

[17] K.M. Singh, J.J.R. Williams, A parallel fictitious domain multigrid preconditioner for the solution of Poisson's equation in complex geometries, Comput. Methods Appl. Mech. Eng. 194 (2005) 4845–4860.

[18] M. Benzi, Preconditioning techniques for large linear systems: A survey, J. Comput. Phys. 182 (2002) 418–477.

[19] B.F. Smith, P.E. Bjørstad, W.D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, New York, 1996.

[20] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform, Proc. 18th Euromicro Conf. Parallel, Distrib. Network-Based Process. (2010) 583–592.

[21] R. Aubry, F. Mut, R. Löhner, J.R. Cebral, Deflated preconditioned conjugate gradient solvers for the pressure–Poisson equation, J. Comput. Phys. 227 (2008) 10196–10208.

[22] A. Gaul, M.H. Gutknecht, J. Liesen, R. Nabben, A framework for deflated and augmented Krylov subspace methods, SIAM J. Matrix Anal. Appl. 34 (2013) 495–518.

[23] M.H. Gutknecht, Deflated and augmented Krylov subspace methods : A framework for deflated BiCG and related solvers, SIAM J. Matrix Anal. Appl. (submitted).

[24] M.H. Gutknecht, Spectral deflation in Krylov solvers: a theory of coordinate space based methods, Electronic Trans. Numer. Anal. 39 (2012)156–185.

[25] B.J. Mohd-Yusof, Development of immersed boundary methods for complex geometry, Annual Research Briefs, Centre for Turbulence Research, Stanford University, 1998, pp. 325–336.

[26] R. Mittal, H. Dong, M. Bozkurttas, F.M. Najjar, A. Vargas, A. von Loebbecke, A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries, J. Comput. Phys. 227 (2008) 4825–4852.

[27] M.-C. Lai, C.S. Peskin, An immersed boundary method with formal second-order accuracy and reduced numerical viscosity, J. Comput. Phys. 160 (2000) 705–719.

[28] J. Kim, D. Kim, H. Choi, An immersed-boundary finite-volume method for simulations of flow in complex geometries, J. Comput. Phys. 171 (2001) 132–150.

[29] J.H. Seo, R. Mittal, A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations, J. Comput. Phys. 230 (2011) 7347–7363.

[30] J. Yang, F. Stern, Sharp interface immersed-boundary/level-set method for wave–body interactions, J. Comput. Phys. 228 (2009) 6590–6616.

[31] T. Ye, R. Mittal, H.S. Udaykumar, W. Shyy, An accurate Cartesian grid method for viscous incompressible flows with complex immersed boundaries, J. Comput. Phys. 156 (1999) 209–240.

[32] J. Yang, E. Balaras, An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries, J. Comput. Phys. 215 (2006) 12–40.

[33] C.S. Wu, D.L. Young, C.L. Chiu, Simulation of wave–structure interaction by hybrid Cartesian/immersed boundary and arbitrary Lagrangian–Eulerian finite-element method, J. Comput. Phys. 254 (2013) 155–183.

[34] B. Yildirim, S. Lin, S. Mathur, J.Y. Murthy, A parallel implementation of fluid–solid interaction solver using an immersed boundary method, Comput. Fluids. 86 (2013) 251–274.

[35] K. Anupindi, Y. Delorme, D. a Shetty, S.H. Frankel, A novel multiblock immersed boundary method for large eddy simulation of complex arterial hemodynamics., J. Comput. Phys. 254 (2013) 200–218.

[36] C. Ji, A. Munjiza, J.J.R. Williams, A novel iterative direct-forcing immersed boundary method and its finite volume applications, J. Comput. Phys. 231 (2012) 1797–1821.

[37] D. Xu, E. Kaliviotis, A. Munjiza, E. Avital, C. Ji, J. Williams, Large scale simulation of red blood cell aggregation in shear flows., J. Biomech. 46 (2013) 1810–7.

[38] X. Bai, E.J. Avital, A. Munjiza, J.J.R. Williams, Numerical simulation of a marine current turbine in free surface flow, Renew. Energy 63 (2014) 715–723.

[39] K.M. Singh, J.J.R. Williams, Application of the additive Schwarz method to large scale Poisson problems, Commun. Numer. Methods Eng. 20 (2004) 193–205.

[40] C.C. Douglas, S. Malhotra, M.H. Schultz, A characterization of mapping unstructured grids onto structured grids and using multigrid as a preconditioner, BIT 37 (1997) 661–677.